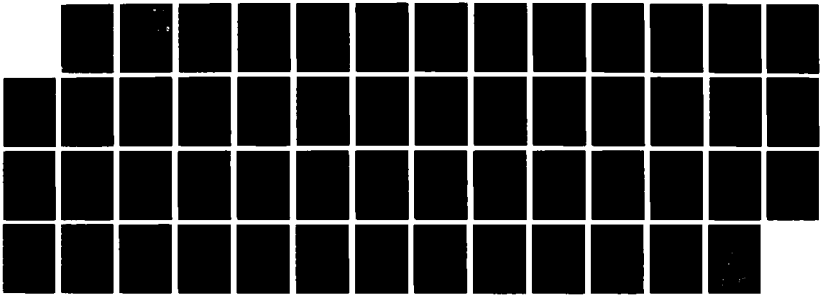
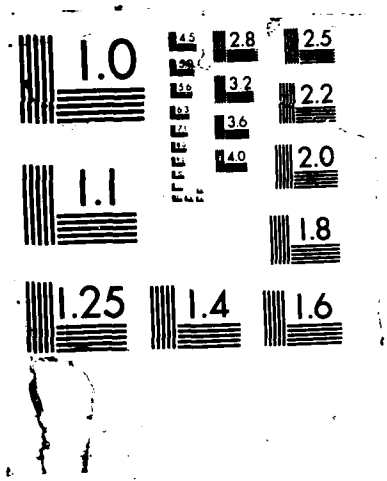
AD-A186 571 ADA (TRADE NAME) COMPILER VALIDATION SUMMARY REPORT SYSTEMS DESIGNERS PLC.. (U) NATIONAL COMPUTING CENTRE LTD MANCHESTER (ENGLAND) 01 DEC 86 1/1

UNCLASSIFIED F/G 12/5 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5

AVF Control Number: AVF-VSR-90502/07

Ada* COMPILER
VALIDATION SUMMARY REPORT:
Systems Designers plc
SD Ada-Plus VAX/VMS x MC68020
Version 2B.00
Host : VAX 8600
Target: Motorola MC68020

Completion of On-Site Testing:
1 December 1986

Prepared By:
National Computing Centre Limited
Oxford Road
Manchester
M1 7ED
UK

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington, D.C.
USA

*Ada is a registered trademark of the United States Government (Ada Joint Program Office).

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETEING FORM
1. REPORT NUMBER | 2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Ada Compiler Validation Summary Report: SD Ada-Plus VAX/VMS x MC68020	5. TYPE OF REPORT & PERIOD COVERED 1 Dec 1986 to 1 Dec 1987 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) The National Computing Centre Ltd.	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS The National Computing Centre Ltd Vony Gwillim Oxford Rd., Manchester, UK	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081ASD/SIOL	12. REPORT DATE 1 Dec 1986 13. NUMBER OF PAGES 47
14. MONITORING AGENCY NAME & ADDRESS(If different from Controlling Office) The National Computing Centre Ltd.	15. SECURITY CLASS (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20. If different from Report)

UNCLASSIFIED

18. SUPPLEMENTARY NOTES

19. KEYWORDS (Continue on reverse side if necessary and identify by block number)

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

See Attached.

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE
1 JAN 73 S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

EXECUTIVE SUMMARY

This Validation Summary Report (VSR) summarizes the results and conclusions of validation testing performed on the SD Ada-Plus VAX/VMS x MC68020, 2B.00, using Version 1.8 of the Ada* Compiler Validation Capability (ACVC). The SD Ada-Plus VAX/VMS x MC68020 is hosted on a VAX 8600 operating under VMS, 4.2. Programs processed by this compiler may be executed on a Motorola MC68020.

On-site testing was performed 28 November 1986 through 1 December 1986 at Systems Designers plc, Camberley, under the direction of The National Computing Centre Ltd (AVF), according to Ada Validation Organization (AVO) policies and procedures. The AVF identified 2102 of the 2399 tests in ACVC Version 1.8 to be processed during on-site testing of the compiler. The 19 tests withdrawn at the time of validation testing, as well as the 278 executable tests that make use of floating-point precision exceeding that supported by the implementation were not processed. After the 2102 tests were processed, results for Class A, C, D, or E tests were examined for correct execution. Compilation listings for Class B tests were analyzed for correct diagnosis of syntax and semantic errors. Compilation and link results of Class L tests were analyzed for correct detection of errors. There were 184 of the processed tests determined to be inapplicable; The remaining 1918 tests were passed.

The results of validation are summarized in the following table:

RESULT	CHAPTER												TOTAL
	2	3	4	5	6	7	8	9	10	11	12	14	
Passed	93	205	280	244	161	97	138	261	123	31	218	67	1918
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	23	120	140	3	0	0	1	1	7	1	0	166	462
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	19
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	2399

The AVF concludes that these results demonstrate acceptable conformity to ANSI/MIL-STD-1815A Ada.

*Ada is a registered trademark of the United States Government (Ada Joint Program Office).

EXECUTIVE SUMMARY

This Validation Summary Report (VSR) summarizes the results and conclusions of validation testing performed on the SD Ada-Plus VAX/VMS x MC68020, 2B.00, using Version 1.8 of the Ada* Compiler Validation Capability (ACVC). The SD Ada-Plus VAX/VMS x MC68020 is hosted on a VAX 8600 operating under VMS, 4.2. Programs processed by this compiler may be executed on a Motorola MC68020.

On-site testing was performed 28 November 1986 through 1 December 1986 at Systems Designers plc, Camberley, under the direction of The National Computing Centre Ltd (AVF), according to Ada Validation Organization (AVO) policies and procedures. The AVF identified 2102 of the 2399 tests in ACVC Version 1.8 to be processed during on-site testing of the compiler. The 19 tests withdrawn at the time of validation testing, as well as the 278 executable tests that make use of floating-point precision exceeding that supported by the implementation were not processed. After the 2102 tests were processed, results for Class A, C, D, or E tests were examined for correct execution. Compilation listings for Class B tests were analyzed for correct diagnosis of syntax and semantic errors. Compilation and link results of Class L tests were analyzed for correct detection of errors. There were 184 of the processed tests determined to be inapplicable; The remaining 1918 tests were passed.

The results of validation are summarized in the following table:

RESULT	CHAPTER												TOTAL
	2	3	4	5	6	7	8	9	10	11	12	14	
Passed	93	205	280	244	161	97	138	261	123	31	218	67	1918
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	23	120	140	3	0	0	1	1	7	1	0	166	462
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	19
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	2399

The AVF concludes that these results demonstrate acceptable conformity to ANSI/MIL-STD-1815A Ada.

*Ada is a registered trademark of the United States Government (Ada Joint Program Office).

Ada* Compiler Validation Summary Report:

Compiler Name: SD Ada-Plus VAX/VMS x MC68020

Host:
VAX 8600 under
VMS
4.2

Target:
Motorola MC68020 under
no operating system

Testing Completed 1 December 1986 Using ACVC 1.8

This report has been reviewed and is approved.

The National Computing Centre Ltd
Vony Gwillim
Oxford Road
Manchester
M1 7ED

Ada Validation Office
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA

Ada Joint Program Office
Virginia L. Castor
Director
Department of Defense
Washington DC

87 1236

*Ada is a registered trademark of the United States Government (Ada Joint Program Office).

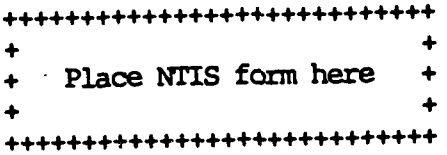

Place NTIS form here

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from characteristics of particular operating systems, hardware, or implementation strategies. All of the dependencies demonstrated during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behaviour that is implementation dependent but permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- To attempt to identify any unsupported language constructs required by the Ada Standard.
- To determine that the implementation-dependent behaviour is allowed by the Ada Standard

Testing of this compiler was conducted by NCC under the direction of the AVF according to policies and procedures established by the Ada Validation Organisation (AVO). On-site testing was conducted from 28 November 1986 through 1 December 1986 at Systems Designers plc., Camberley.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. 552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organisations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:

Ada Validation Facility
The National Computing Centre Ltd
Oxford Road
Manchester
M1 7ED
United Kingdom

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, FEB 1983.

2. Ada Validation Organization: Policies and Procedures, MITRE Corporation, JUN 1982, PB 83-110601.

3. Ada Compiler Validation Capability Implementer's Guide, SofTech, Inc., DEC 1984.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. A set of programs that evaluates the conformity of a compiler to the Ada language specification, ANSI/MIL-STD-1815A.
Ada Standard	ANSI/MIL-STD-1815A, February 1983.
Applicant	The agency requesting validation.
AVF	The National Computing Centre Ltd. In the context of this report, the AVF is responsible for conducting compiler validations according to established policies and procedures.
AVO	The Ada Validation Organization. In the context of this report, the AVO is responsible for setting procedures for compiler validations.
Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	A test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.

Host — The computer on which the compiler resides.

Inapplicable test — A test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.

Passed test — A test for which a compiler generates the expected result.

Target — The computer for which a compiler generates code.

Test — A program that checks a compiler's conformity regarding a particular feature or features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.

Withdrawn — A test found to be incorrect and not used to check conformity to test the Ada language specification. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce link errors.

Class A tests check that legal Ada programs can be successfully compiled and executed. However, no checks are performed during execution to see if the test objective has been met. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capabilities of a compiler. Since there are no requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimization allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard. The operation of these units is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of the tests in the ACVC follow conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation are listed in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: SD Ada-Plus VAX/VMS x MC68020

ACVC Version: 1.8

Certification Expiration Date: 17 December 1987

Host Computer:

Machine :	VAX 8600
Operating System:	VMS 4.2
Memory Size:	20 M byte

Target Computer:

Machine :	Motorola MC68020 implemented on Motorola MVME 133 board, incorporating MC68881 floating point co-processor.
Operating System:	no operating system
Memory Size:	1 M byte

Communications Network:	RS232C connector via a null modem using a protocol conforming to RS232C.

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behaviour of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. This compiler is characterized by the following interpretations of the Ada Standard:

- Capacities.

  The compiler correctly processes compilations containing loop statements nested to 65 levels, block statements nested to 65 levels, and recursive procedures separately compiled as subunits nested to 17 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See tests D55A03A..H (8 tests), D56001B, D64005E..G (3 tests), and D29002K.)

- Universal integer calculations.

  An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX_INT. This implementation does not reject such calculations and processes them correctly. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

- Predefined types.

  This implementation supports the additional predefined type SHORT_INTEGER in the package STANDARD. (See tests B86001C and B86001D.)

- Based literals.

  An implementation is allowed to reject a based literal with a value exceeding SYSTEM.MAX_INT during compilation, or it may raise NUMERIC_ERROR or CONSTRAINT_ERROR during execution. This implementation raises NUMERIC_ERROR during execution. (See test E24101A.)

- Array Types.

  An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/ or SYSTEM.MAX_INT.

  A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises NUMERIC_ERROR when the array type is declared. (See test C52103X.)

A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises NUMERIC_ERROR when the array type is declared. (See test C52104Y.)

A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises NUMERIC_ERROR when the array type is declared. (See test E52103Y.)

In assigning one-dimensional array types, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. In assigning two-dimensional array types, the expression does not appear to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

- Discriminated types.

  During compilation, an implementation is allowed to either accept or reject an incomplete type with discriminants that is used in an access type definition with a compatible discriminant constraint. This implementation accepts such subtype indications. (See test E38104A.)

  In assigning record types with discriminants, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

- Aggregates.

  In the evaluation of a multi-dimensional aggregate, all choices appear to be evaluated before checking against the index type. (See tests C43207A and C43207B.)

  In the evaluation of an aggregate containing subaggregates, all choices are evaluated before being checked for identical bounds. (See test E43212B.)

  All choices are evaluated before CONSTRAINT_ERROR is raised if a bound in a nonnull range of a nonnull aggregate does not belong to an index subtype. (See test E43211B.)

. Functions

An implementation may allow the declaration of a parameterless function and an enumeration literal having the same profile in the same immediate scope, or it may reject the function declaration. If it accepts the function declarations, the use of the enumeration literal's identifier denotes the function. This implementation rejects the declarations. (See test E66001D.)

. Representation clauses.

The Ada Standard does not require an implementation to support representation clauses. If a representation clause is not supported, then the implementation must reject it. While the operation of representation clauses is not checked by Version 1.8 of the ACVC, they are used in testing other language features. This implementation accepts 'SIZE and 'STORAGE_SIZE for tasks, 'STORAGE_SIZE for collections, and 'SMALL clauses. Enumeration representation clauses, including those that specify noncontiguous values, appear to be supported. (See tests C55B16A, C87B62A, C87B62B, C87B62C, and BC1002A.)

. Pragmas.

The pragma INLINE is not supported for procedures. The pragma INLINE is not supported for functions. (See tests CA3004E and CA3004F.)

. Input/Output.

The package SEQUENTIAL_IO can be instantiated with unconstrained array types and record types with discriminants. The package DIRECT_IO can be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, AE2101H, CE2201D, CE2201E, and CE2401D.)

This implementation implements input/output packages SEQUENTIAL_IO, DIRECT_IO and TEXT_IO as "null" packages. The package raises two possible exceptions, details of which are given in paragraph F.8 of Appendix B.

. Generics.

Generic subprogram declarations and bodies can be compiled in separate compilations. (See test CA2009F.)

Generic package declarations and bodies can be compiled in separate compilations. (See tests CA2009C and BC3205D.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.8 of the ACVC contains 2399 tests. When validation testing of SD Ada-Plus VAX/VMS x MC68020 was performed, 19 tests had been withdrawn. The remaining 2380 tests were potentially applicable to this validation. The AVF determined that 462 tests were inapplicable to this implementation, and that the 1918 applicable tests were passed by the implementation.

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	69	865	912	17	11	44	1918
Failed	0	0	0	0	0	0	0
Inapplicable	0	2	456	0	2	2	462
Withdrawn	0	7	12	0	0	0	19
TOTAL	69	874	1380	17	13	46	2399

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER												
	2	3	4	5	6	7	8	9	10	11	12	14	TOTAL
Passed	93	205	280	244	161	97	138	261	123	31	218	67	1918
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	23	120	140	3	0	0	1	1	7	1	0	166	462
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	19
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	2399

3.4 WITHDRAWN TESTS

The following 19 tests were withdrawn from ACVC Version 1.8 at the time of this validation:

C32114A	C41404A	B74101B
B33203C	B45116A	C87B50A
C34018A	C48008A	C92005A
C35904A	B49006A	C940ACA
B37401A	B4A010C	CA3005A..D (4 tests)
		BC3204C

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. For this validation attempt, 462 tests were inapplicable for the reasons indicated:

- C34001E, B52004D, B55B09C, and C55B07A use LONG_INTEGER which is not supported by this compiler.
- C34001F and C35702A use SHORT_FLOAT which is not supported by this compiler.
- C34001G and C35702B use LONG_FLOAT which is not supported by this compiler.

- C64104M, CB1010B, CZ1201D requires storage space for a fixed size collection which is exceeded during execution. On the MC68020 target computer the default collection size allocation is 1K bytes. STORAGE_ERROR is raised during execution because the total size of the objects within the collection is greater than this default storage size. Although these three tests were ruled inapplicable, modified versions using representation clauses to increase the collection sizes for C64104M, CB1010B and CZ1201D to 4K, 10K and 2K respectively. These modified tests all executed successfully.

- B86001DT requires a predefined numeric type other than those defined by the Ada language in package STANDARD. There is no such type for this implementation.

- C86001F. A separate package is used to collect the executable test results from the MC68020 target. The package TEST_IO uses the package SYSTEM, thus when this test recompiles package SYSTEM it invalidates the package TEST_IO. This means that the test cannot be built and executed.

- C96005B checks implementations for which the smallest and largest values in type DURATION are different from the smallest and largest values in DURATION's base type. This is not the case for this implementation.

- CA3004E, EA3004C, and LA3004A use INLINE pragma for procedures which is not supported by this compiler.

- CA3004F, EA3004D, and LA3004B use INLINE pragma for functions which is not supported by this compiler.

- This implementation raises USE_ERROR when an attempt is made to create/open a file. As a result, the following 166 tests are inapplicable, as is CZ1103A (one of the support units), although this test does not appear in the counts.

CE2102D..F (3 tests)	CE2204A..B (2 tests)	CE3104A
CE2102I..J (2 tests)	CE2210A	CE3107A
CE2104A..D (4 tests)	CE2401A..F (6 tests)	CE3108A..B (2 tests)
CE2105A	CE2404A	CE3109A
CE2106A	CE2405B	CE3110A
CE2107A..F (6 tests)	CE2406A	CE3111A..E (5 tests)
CE2108A..D (4 tests)	CE2407A	CE3112A..B (2 tests)
CE2109A	CE2408A	CE3114A..B (2 tests)
CE2110A..C (3 tests)	CE2409A	CE3115A
CE2111A..E (5 tests)	CE2410A	CE3203A
CE2111G..H (2 tests)	CE3102B	CE3208A
CE2201A..F (6 tests)	CE3103A	CE3310A..C (3 tests)

CE3302A	CE3410C..F (4 tests)	CE3704M..O (3 tests)
CE3305A	CE3411A	CE3706D..F (2 tests)
CE3402A..D (4 tests)	CE3412A	CE3804A..E (5 tests)
CE3403A..C (3 tests)	CE3413A	CE3804G ,I (2 tests)
CE3403E..F (2 tests)	CE3413C	CE3804M
CE3404A..C (3 tests)	CE3602A..D (4 tests)	CE3805A..B (2 tests)
CE3405A..D (4 tests)	CE3603A	CE3806A
CE3406A..D (4 tests)	CE3604A	CE3806D..E (2 tests)
CE3407A..C (3 tests)	CE3605A..E (5 tests)	CE3905A..C (3 tests)
CE3408A..C (3 tests)	CE3606A..B (2 tests)	CE3905L
CE3409A	CE3704A..B (2 tests)	CE3906A..C (3 tests)
CE3409C..F (4 tests)	CE3704D..F (3 tests)	CE3906E..F (2 tests)
CE3410A		

- The following 278 tests make use of floating-point precision that exceeds the maximum of 6 supported by the implementation:

C24113C..Y (23 tests)
C35705C..Y (23 tests)
C35706C..Y (23 tests)
C35707C..Y (23 tests)
C35708C..Y (23 tests)
C35802C..Y (23 tests)
C45241C..Y (23 tests)
C45321C..Y (23 tests)
C45421C..Y (23 tests)
C45424C..Y (23 tests)
C45521C..Z (24 tests)
C45621C..Z (24 tests)

Also one of the support tests, CZ1103A does not produce output equivalent to the expected output. This is because the exception USE_ERROR is raised on all attempts to create a file within this test.

3.6 SPLIT TESTS

If one or more errors do not appear to have been detected in a Class B test because of compiler error recovery, then the test is split into a set of smaller tests that contain the undetected errors. These splits are then compiled and examined. The splitting process continues until all errors are detected by the compiler or until there is exactly one error per split. Any Class A, Class C, or Class E test that cannot be compiled and executed because of its size is split into a set of smaller subsets that can be processed.

Splits were required for 6 Class B tests.

B22003A	B74401C	BC1202E
B29001A	BC10AEB	BC3204B

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.8 produced by SD Ada-Plus VAX/VMS x MC68020 was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behaviour on all inapplicable tests.

3.7.2 Test Method

Testing of SD Ada-Plus VAX/VMS x MC68020 using ACVC Version 1.8 was conducted on-site by a validation team from the AVF. The configuration consisted of a VAX 8600 host operating under VMS, 4.2, and a Motorola MC68020 target under no operating system. The host and target computers were linked via RS232C connector.

A magnetic tape containing all tests was taken on-site by the validation team for processing. The magnetic tape contained tests that make use of implementation-specific values which were customized before being written to the magnetic tape. Tests requiring splits during the prevalidation testing were not included in their split form on the magnetic tape.

The contents of the magnetic tape were loaded directly onto the host computer.

After the test files were loaded to disk, the full set of tests was compiled and linked on the VAX 8600, and all executable tests were run on the Motorola MC68020. Object files were linked on the host computer, and executable images were transferred to the target computer via RS232C connector. Results were printed from the host computer, with results being transferred to the host computer via RS232C connector.

The compiler was tested using command scripts provided by Systems Designers plc. and reviewed by the validation team. The following options were in effect for testing:

Option	Effect
"list=>on"	this ensures that the compilation listings produced by the compiler contain a full listing of the test source.

Tests were compiled, linked and executed (as appropriate) using a single host computer and a single target computer. Test output, compilation listings, and job logs were captured on magnetic tape and archived at AVF. The listings examined on-site by the validation team were also archived.

3.7.3 TEST SITE

The validation team arrived at Systems Designers plc., Camberley on 28 November 1986 and departed after testing was completed on 1 December 1986.

APPENDIX A

COMPLIANCE STATEMENT

Systems Designers plc., has submitted the following compliance statement concerning the SD Ada-Plus VAX/VMS x MC68020.

Compliance Statement

Base Configuration:

Compiler: SD Ada-Plus VAX/VMS x MC68020, 2B.00

Test Suite: Ada* Compiler Validation Capability, Version 1.8

Host Computer:

Machine:	VAX 8600
Operating System:	VMS 4.2

Target Computer:

Machine:	Motorola MC68020 implemented on Motorola MVME 133 board, incorporating MC68881 floating point co-processor
Operating System:	no operating system
Communications Network:	RS232C connector via a null modem using a protocol conforming to RS232C.

Systems Designers plc. has made no deliberate extensions to the Ada language standard.

Systems Designers plc. agrees to the public disclosure of this report.

Systems Designers plc. agrees to comply with the Ada trademark policy, as defined by the Ada Joint Program Office.

Date: 1 December 1986

Systems Designers plc
Bill Davison
Customer Services Manager

*Ada is registered trademark of the United States Government (Ada Joint Program Office).

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of MIL-STD-1815A, and to certain allowed restrictions on representation classes. The implementation-dependent characteristics of the SD Ada-Plus VAX/VMS x MC68020, 2B.00 are described in the following sections which discuss topics one through eight as stated in Appendix F of the Ada Language Reference Manual (ANSI/MIL-STD-1815A). The specification of the package STANDARD is also included in this appendix.

SYSTEMS DESIGNERS

Ada(R) -Plus

VAX/VMS x MC68020

APPENDIX F TO THE REFERENCE MANUAL

Systems Designers plc,
Pembroke House,
Pembroke Broadway,
Camberley,
Surrey.
GU15 3XD
UNITED KINGDOM

Telephone: 0276 686200
Telex : 858280 SYSDES G

D.A.REF.AF[BC-MH]

Issue 1.0

December 1986

AMENDMENT RECORD

Amendment Notification	Date of Issue	Incorporated By	Date Incorporated

CONTENTS

FIGURES

PREFACE

This document describes the implementation-dependent characteristics of the VAX/VMS x MC68020 SD-Ada Compiler.

The document should be considered as Appendix F of the Reference Manual for the Ada Programming Language.

APPENDIX F

IMPLEMENTATION-DEPENDENT CHARACTERISTICS

F.1 IMPLEMENTATION-DEPENDENT PRAGMAS

F.1.1 Pragma EXPORT

Form

```
pragma EXPORT ([ADA_NAME=>] simple_name,
               [EXT_NAME=>] "name_string");
```

The pragma EXPORT takes the name of an Ada variable in the first parameter position and a string in the second parameter position. The name must be the simple name of a variable in the package level static data area in scope, and name-string must be a string literal which is unique in any program produced for the target, otherwise the program is erroneous.

The parameter name_string must be a string literal which conforms to the naming conventions imposed by the MC68020 builder. The name must be no more than eight characters in length and start with a dot or upper case letter. The rest of the characters are restricted to being a digit, dot, dollar, underline or upper case letter.

Position

The pragma EXPORT may be placed at the position of a basic declarative item of a library package specification or in the declarative part of a library package body.

Effect

Use of this pragma causes the compiler to generate additional linkage information. This associates the string literal of the second parameter with the variable nominated by the first parameter. This external naming facility is restricted to data objects held in static areas.

F.1.2 Pragma DEBUG

Form

```
pragma DEBUG ([NAME=>]name);
```

The pragma DEBUG takes a name as the single argument. The value yielded by the parameter must be scalar or access type.

Position

The pragma DEBUG may be placed at the position of a basic_declarative_item or a statement where the name is in scope.

Effect

Use of this pragma causes the compiler to generate tracing code, and auxiliary information in debug symbol tables. This tracing code is loaded into the target computer in such a way that the main thread of normal execution perceives no reference to the trace code, and the values embedded in the main thread code, such as offsets, remain unaffected.

The tracing code may be activated by use of the Debug System.

F.1.3 Pragma SUPPRESS_ALL

Form

pragma SUPPRESS_ALL;

This pragma has no parameters.

Position

The pragma SUPPRESS_ALL is only allowed at the start of a compilation before the first compilation unit.

Effect

Use of this pragma prevents the compiler from generating any run-time checks for CONSTRAINT_ERROR or NUMERIC_ERROR.

F.2 IMPLEMENTATION-DEPENDENT ATTRIBUTES

There are no such attributes.

F.3 PACKAGE SYSTEM

The specification of the package SYSTEM is given in Figure F.1.

In order to obtain addresses the routine CONVERT_ADDRESS is supplied. The function takes a parameter of type EXTERNAL_ADDRESS which must be 8 or less Hexadecimal characters representing an address. If the address is outside the range 0..MEMORY_SIZE-1 the predefined exception CONSTRAINT_ERROR is raised. CONSTRAINT_ERROR is also raised if the EXTERNAL_ADDRESS contains any non-hexadecimal characters.

The function is overloaded to take a parameter of type ADDRESS and return EXTERNAL_ADDRESS. This value will have all leading zeros suppressed unless the address is zero in which case a single zero will be returned.

```
package SYSTEM  is

   type ADDRESS is private

   type NAME    is (MC68020);

   SYSTEM_NAME   : constant NAME := MC68020;
   STORAGE_UNIT  : constant      := 8;
   MEMORY_SIZE   : constant      := 2**32;
   MIN_INT       : constant      := -(2**31);
   MAX_INT       : constant      := (2**31)-1;
   MAX_DIGITS    : constant      := 6;
   MAX_MANTISSA  : constant      := 31;
   FINE_DELTA    : constant      := 2#1.0#E-30;
   TICK          : constant      := 2#1.0#E-7;

   subtype PRIORITY is INTEGER range 0 .. 15;

   type UNIVERSAL_INTEGER is range MIN_INT .. MAX_INT;

   subtype EXTERNAL_ADDRESS is STRING;

   function   CONVERT_ADDRESS (ADDR   : EXTERNAL_ADDRESS)
       return ADDRESS;

   function   CONVERT_ADDRESS (ADDR   : ADDRESS)
       return EXTERNAL_ADDRESS;

   function   "+"              (ADDR   : ADDRESS;
                                OFFSET : UNIVERSAL_INTEGER)
       return ADDRESS;

   private

       -- type ADDRESS is system-dependent

   end SYSTEM;
```

Figure F.1

Package SYSTEM

F.4 RESTRICTIONS ON REPRESENTATION CLAUSES

F.4.1 Length Clauses

F.4.1.1 Attribute SIZE

The value specified for SIZE must not be less than that chosen by default by the compiler (e.g. 8 for enumeration types, 32 for integer types, real types and access types, etc.). The value given is ignored.

F.4.1.2 Attribute STORAGE_SIZE

For access types the limit is governed by the indexing range of the target machine and the maximum is equivalent to SYSTEM.ADDRESS'LAST.

For task types the limit is also SYSTEM.ADDRESS'LAST.

F.4.1.3 Attribute SMALL

Only values which are powers of two are supported for this attribute.

F.4.2 Record Representation Clauses

F.4.2.1 Alignment Clause

The static_simple_expression used to align records onto storage unit boundaries must deliver the values 1 or 2.

F.4.2.2 Component Clause

The static range is restricted to ranges which force component alignment onto storage unit boundaries only, (i.e. multiples of 8 bits).

The component size defined by the static range must not be less than the minimum number of bits required to hold every allowable value of the component. For a component of non-scalar type, the size must not be larger than that chosen by the compiler for the type.

F.4.3 Address Clause

F.4.3.1 Object Addresses

For objects with an address clause, a pointer is declared which points to the object at the given address. There is a restriction however that the object cannot be initialised either explicitly or implicitly (i.e the object cannot be an access type).

F.4.3.2 Entry Addresses

Address clauses for entries are supported; the address given is the address of an interrupt vector.

F.5 IMPLEMENTATION-GENERATED NAMES

There are no implementation-generated names denoting implementation-dependent components.

F.6 INTERPRETATION OF EXPRESSIONS IN ADDRESS CLAUSES

The expressions in an address clause are interpreted as absolute addresses on the target.

F.7 UNCHECKED CONVERSIONS

The implementation imposes the restriction on the use of the generic function UNCHECKED_CONVERSION that the size of the target type must not be greater than the size of the source type.

F.8 CHARACTERISTICS OF THE INPUT/OUTPUT PACKAGES

Packages SEQUENTIAL_IO, DIRECT_IO and the predefined input/output package TEXT_IO are implemented as "null" packages which conform to the specification given in the Ada Language Reference Manual. This package raises the exceptions specified in Chapter 14 of the Language Reference Manual. There are two possible exceptions which are raised by this package. These are given here in the order in which they will be raised.

a) The exception STATUS_ERROR is raised by an attempt to operate upon a file that is not open (no files can be opened).

b) The exception USE_ERROR is raised if exception STATUS_ERROR is not raised.

Note that MODE_ERROR cannot be raised as no file can be opened (therefore it cannot have a current mode) and NAME_ERROR cannot be raised since there are no restrictions on file names.

The predefined package IO_EXCEPTIONS is defined in the Ada Language Reference Manual.

The predefined package LOW_LEVEL_IO is not provided.

The implementation-dependent characteristics are described in Sections F.8.1 to F.8.2.

F.8.1 The Package TEXT_IO

When any procedure is called the exception STATUS_ERROR or USE_ERROR is raised (there are no restrictions on the format of the NAME or FORM parameters).

The type COUNT is defined:-

```
type COUNT is range 0 .. INTEGER'LAST;
```

and the subtype FIELD is defined:

```
subtype FIELD is INTEGER range 0 .. 132;
```

F.8.2 The Package IO_EXCEPTIONS

The specification of the package is the same as that given in the Ada Language Reference Manual.

F.9 PACKAGE STANDARD

The specification of package STANDARD is given in Figure F.2.

```
package STANDARD is

  type BOOLEAN is (FALSE, TRUE);

  type SHORT_INTEGER is range -32768 .. 32767;

  type INTEGER is range
    - 2147483648 .. 2147483647;

  type FLOAT is digits 6 range
    - 16#0.FFFFFF#E32 .. 16#0.FFFFFF#E32;

   type CHARACTER is

     (nul, soh, stx, etx,      eot, enq, ack, bel,

      bs , ht , lf , vt ,      ff , cr , so , si ,

      dle, dc1, dc2, dc3,      dc4, nak, syn, etb,

      can, em , sub, esc,      fs , gs , rs , us ,

      ' ', '!', '"', '#',      '$', '%', '&', ''',
      '(', ')', '*', '+',      ',', '-', '.', '/',
      '0', '1', '2', '3',      '4', '5', '6', '7',
      '8', '9', ':', ';',      '<', '=', '>', '?',

      '@', 'A', 'B', 'C',      'D', 'E', 'F', 'G',
      'H', 'I', 'J', 'K',      'L', 'M', 'N', 'O',
      'P', 'Q', 'R', 'S',      'T', 'U', 'V', 'W',
      'X', 'Y', 'Z', '[',      '\', ']', '^', '_',

      '`', 'a', 'b', 'c',      'd', 'e', 'f', 'g',
      'h', 'i', 'j', 'k',      'l', 'm', 'n', 'o',
      'p', 'q', 'r', 's',      't', 'u', 'v', 'w',
      'x', 'y', 'z', '{',      '|', '}', '~', del);
```

Figure F.2 (1 of 4)

Package STANDARD

```
for   CHARACTER use   -- ASCII characters without holes
   (0   , 1   , 2   , 3   , 4   , 5   , 6   , 7   ,
    8   , 9   , 10  , 11  , 12  , 13  , 14  , 15  ,
    16  , 17  , 18  , 19  , 20  , 21  , 22  , 23  ,
    24  , 25  , 26  , 27  , 28  , 29  , 30  , 31  ,
    32  , 33  , 34  , 35  , 36  , 37  , 38  , 39  ,
    40  , 41  , 42  , 43  , 44  , 45  , 46  , 47  ,
    48  , 49  , 50  , 51  , 52  , 53  , 54  , 55  ,
    56  , 57  , 58  , 59  , 60  , 61  , 62  , 63  ,
    64  , 65  , 66  , 67  , 68  , 69  , 70  , 71  ,
    72  , 73  , 74  , 75  , 76  , 77  , 78  , 79  ,
    80  , 81  , 82  , 83  , 84  , 85  , 86  , 87  ,
    88  , 89  , 90  , 91  , 92  , 93  , 94  , 95  ,
    96  , 97  , 98  , 99  , 100, 101, 102, 103,
    104, 105, 106, 107, 108, 109, 110, 111,
    112, 113, 114, 115, 116, 117, 118, 119,
    120, 121, 122, 123, 124, 125, 126, 127);

package ASCII is

   -- Control characters:

   NUL         : constant CHARACTER := nul;
   SOH         : constant CHARACTER := soh;
   STX         : constant CHARACTER := stx;
   ETX         : constant CHARACTER := etx;
   EOT         : constant CHARACTER := eot;
   ENQ         : constant CHARACTER := enq;
   ACK         : constant CHARACTER := ack;
   BEL         : constant CHARACTER := bel;
   BS          : constant CHARACTER := bs;
   HT          : constant CHARACTER := ht;
   LF          : constant CHARACTER := lf;

   VT          : constant CHARACTER := vt;
   FF          : constant CHARACTER := ff;
   CR          : constant CHARACTER := cr;
   SO          : constant CHARACTER := so;
   SI          : constant CHARACTER := si;
   DLE         : constant CHARACTER := dle;
   DC1         : constant CHARACTER := dc1;
   DC2         : constant CHARACTER := dc2;
   DC3         : constant CHARACTER := dc3;
   DC4         : constant CHARACTER := dc4;
   NAK         : constant CHARACTER := nak;
   SYN         : constant CHARACTER := syn;
```

Figure F.2 (2 of 4)

Package STANDARD

```
ETB         : constant CHARACTER := etb;
CAN         : constant CHARACTER := can;
EM          : constant CHARACTER := em;
SUB         : constant CHARACTER := sub;
ESC         : constant CHARACTER := esc;
FS          : constant CHARACTER := fs;
GS          : constant CHARACTER := gs;
RS          : constant CHARACTER := rs;
US          : constant CHARACTER := us;
DEL         : constant CHARACTER := del;

-- Other characters:

EXCLAM      : constant CHARACTER := '!';
QUOTATION   : constant CHARACTER := '"';
SHARP       : constant CHARACTER := '#';
DOLLAR      : constant CHARACTER := '$';
PERCENT     : constant CHARACTER := '%';
AMPERSAND   : constant CHARACTER := '&';
COLON       : constant CHARACTER := ':';
SEMICOLON   : constant CHARACTER := ';';
QUERY       : constant CHARACTER := '?';
AT_SIGN     : constant CHARACTER := '@';

L_BRACKET   : constant CHARACTER := '[';
BACK_SLASH  : constant CHARACTER := '\';
R_BRACKET   : constant CHARACTER := ']';
CIRCUMFLEX  : constant CHARACTER := '^';
UNDERLINE   : constant CHARACTER := '_';
GRAVE       : constant CHARACTER := '`';
L_BRACE     : constant CHARACTER := '{';
BAR         : constant CHARACTER := '|';
R_BRACE     : constant CHARACTER := '}';
TILDE       : constant CHARACTER := '~';

-- Lower case letters:

LC_A        : constant CHARACTER := 'a';
LC_B        : constant CHARACTER := 'b';
LC_C        : constant CHARACTER := 'c';
LC_D        : constant CHARACTER := 'd';
LC_E        : cosntant CHARACTER := 'e';
LC_F        : constant CHARACTER := 'f';
LC_G        : constant CHARACTER := 'g';
LC_H        : constant CHARACTER := 'h';
```

Figure F.2 (3 of 4)

Package STANDARD

```
      LC_I        : constant CHARACTER := 'i';
      LC_J        : constant CHARACTER := 'j';
      LC_K        : constant CHARACTER := 'k';
      LC_L        : constant CHARACTER := 'l';
      LC_M        : constant CHARACTER := 'm';
      LC_N        : constant CHARACTER := 'n';
      LC_O        : constant CHARACTER := 'o';
      LC_P        : constant CHARACTER := 'p';
      LC_Q        : constant CHARACTER := 'q';
      LC_R        : constant CHARACTER := 'r';
      LC_S        : constant CHARACTER := 's';
      LC_T        : constant CHARACTER := 't';
      LC_U        : constant CHARACTER := 'u';
      LC_V        : constant CHARACTER := 'v';
      LC_W        : constant CHARACTER := 'w';
      LC_X        : constant CHARACTER := 'x';
      LC_Y        : constant CHARACTER := 'y';
      LC_Z        : constant CHARACTER := 'z';

   end ASCII;

   -- Predefined subtypes:

   subtype NATURAL is INTEGER
                        range 0 .. INTEGER'LAST;

   subtype POSITIVE is INTEGER
                        range 1 .. INTEGER'LAST;

   -- Predefined string type:

   type STRING is array (POSITIVE range <>)
                  of CHARACTER;

   type DURATION is delta 2#1.0#E-7
                    range -16777216.0 .. 16777215.0;

   -- The predefined exceptions:

   CONSTRAINT_ERROR : exception;
   NUMERIC_ERROR    : exception;
   PROGRAM_ERROR    : exception;
   STORAGE_ERROR    : exception;
   TASKING_ERROR    : exception;

end STANDARD;
```

Figure F.2 (4 of 4)

Package STANDARD

F.10 PACKAGE MACHINE_CODE

Package MACHINE_CODE is not supported by the SD-Ada Compiler.

F.11 LANGUAGE-DEFINED PRAGMAS

The definition of certain language-defined pragmas is incomplete in the Ada Language Reference Manual. The implementation restrictions imposed on the use of such pragmas are specified in Sections F.11.1 to F.11.4.

F.11.1 Pragma INLINE

This pragma supplies a recommendation for inline expansion of a subprogram to the compiler. This pragma is ignored by the SD-Ada Compiler.

F.11.2 Pragma INTERFACE

This pragma allows subprograms written in another language to be called from Ada. The SD-Ada Compiler only supports pragma INTERFACE for the language ASSEMBLER. Normal Ada calling conventions are used by the SD-Ada Compiler when generating a call to an ASSEMBLER subprogram.

F.11.2.1 Assembler Names

The name of an interface routine must conform to the naming conventions both of Ada and of the MC68020 builder.

F.11.2.2 Parameter Passing Conventions

Parameters are passed to the called procedure in the order given in the specification of the subprogram, with default expressions evaluated, if present.

Scalars are passed by copy for all parameter modes (the value is copied out for parameters with mode out).

Composite types are passed by reference for all parameter modes.

F.11.2.3 Procedure-Calling Mechanism

The procedure-calling mechanism uses the run-time stack organisation shown in Figure F.3 and the routine entry and exit code shown in Figure F.4.

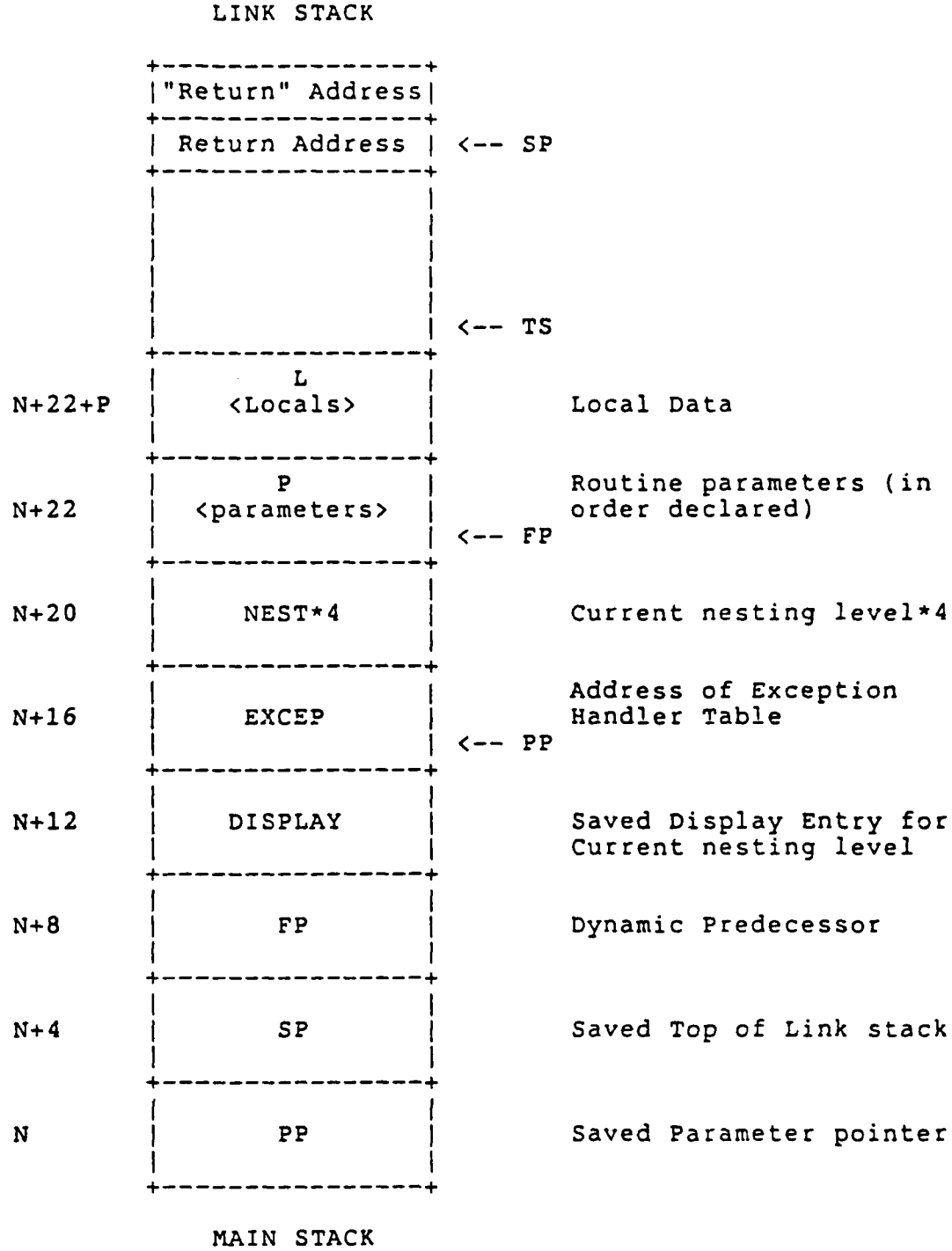


Figure F.3

Routine Activation Record
on Entry to Called Subprogram

The implementation uses the following dedicated and temporary registers:

SP	-	Link Stack Pointer	A7
FP	-	Frame Pointer	A2
PP	-	Parameter Frame Pointer	A3
DP	-	Display Pointer	A0
TS	-	Main Stack Pointer	A1

Macros RM_P_BEGIN and RM_P_END are provided for the routine entry and exit code respectively. This code is shown in Figure F.4.

Routine Entry Code

```
MOVE.L    SP,(PP)+
MOVE.L    FP,(PP)+
MOVE.L    n(DP),(PP)+
MOVEA.L   PP,FP
MOVE.L    FP,(FP)+
MOVE.W    #<nest*4>,(FP)+
MOVE.L    FP,n(DP)
```

Routine Exit Code

```
MOVE.L    -(PP),n(DP)
MOVEA.L   -(PP),FP
MOVEA.L   -(PP),SP
RTS
```

Figure F.4

Routine Entry And Exit Code

F.11.3 Pragma OPTIMISE

This pragma supplies a recommendation to the compiler for the criterion upon which optimisation is to be performed. This pragma is ignored by the SD-Ada Compiler.

F.11.4 Pragma SUPPRESS

This pragma gives permission for specified run-time checks to be omitted by the compiler. This pragma is ignored by the SD-Ada Compiler.

READERS COMMENTS

Do you find this document suitable to your needs? Is it understandable, usable, and well structured? Does it fit appropriately into the Documentation Set which accompanies your Product?

We would like your comments:

Did you find specific errors in the document? If so, can you please submit a User Documentation Problem (UDOP) Report, an example of which is included as part of the Release Details supplied with your product.

Name
Position
Company
Address
Date
Software Version No.

Please return your comments to:

Customer Services Group,
Systems Designers plc,
Pembroke House,
Pembroke Broadway,
Camberley,
Surrey.
GU15 3XD
UNITED KINGDOM

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are identified by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

NAME AND MEANING	VALUE
$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	A....A1 |----| 254 characters
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	A....A2 |----| 254 characters
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	A....A3A....A |----| |----| 127 127 characters
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	A....A4A....A |----| |----| 127 127 characters
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that is is the size of the maximum line length.	0....0298 |----| 252 characters
$BIG_REAL_LIT A real literal that can be either of floating- or fixed-point type, has value of 690.0, and has enough leading zeroes to be the size of the maximum line length.	0....069.0E1 |----| 249 characters

NAME AND MEANING	VALUE
$BLANKS A sequence of blanks twenty characters fewer than the size of the maximum line length.	235 blanks
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2147483647
$EXTENDED_ASCII_CHARS A string literal containing all the ASCII characters with printable graphics that are not in the basic 55 Ada character set.	"abcdefghijklmnopqrstuvwxyz !$%?@[\]^`{}~"
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST	255
$FILE_NAME_WITH_BAD_CHARS An illegal external file name that either contains invalid characters or is too long if no invalid characters exist.	X}]!.dat
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character or is too long if no wild card characters exists.	file*.dat
$GREATER_THAN_DURATION A universal real value that lies between DURATION'BASE'LAST and DURATION'LAST if any, otherwise any value in in the range of DURATION.	2.0
$GREATER_THAN_DURATION_BASE_LAST The universal real value that is greater than DURATION'BASE'LAST, if such a value exists.	16777216.0
$ILLEGAL_EXTERNAL_FILE_NAME1 An illegal external file name.	bad_char^

NAME AND MEANING	VALUE
$ILLEGAL_EXTERNAL_FILE_NAME2 An illegal external file name that is different from $ILLEGAL_EXTERNAL_FILE_NAME1.	bad_char*
$INTEGER_FIRST The universal integer literal expression whose value is INTEGER'FIRST.	-2147483648
$INTEGER_LAST The universal integer literal expression whose value is INTEGER'LAST.	2147483647
$LESS_THAN_DURATION A universal real value that lies between DURATION'BASE'FIRST and DURATION'FIRST if any, otherwise any value in the range of DURATION.	-2.0
$LESS_THAN_DURATION_BASE_FIRST The universal real value that is less than DURATION'BASE'FIRST, if such a value exists.	-16777216.0
$MAX_DIGITS The universal integer literal whose value is the maximum digits supported for floating-point types.	6
$MAX_IN_LEN The universal integer literal whose value is the maximum input line length permitted by the implementation.	20
$MAX_INT The universal integer literal whose value is SYSTEM.MAX_INT.	2147483647
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER if one exists, otherwise any undefined name.	$NAME

NAME AND MEANING	VALUE
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFFFFFE#
$NON_ASCII_CHAR_TYPE An enumerated type definition for a character type whose literals are the identifier NON_NULL and all non_ASCII characters with printable graphics.	(NON_NULL)

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 19 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form "AI-ddddd" is to an Ada Commentary.

. C32114A: An unterminated string literal occurs at line 62.

. B33203C: The reserved word "IS" is misspelled at line 45.

. C34018A: The call of function G at line 114 is ambiguous in the presence of implicit conversions.

. C35904A: The elaboration of subtype declarations SFX3 and SFX4 may raise NUMERIC_ERROR instead of CONSTRAINT_ERROR as expected in the test.

. B37401A: The object declarations at lines 126 through 135 follow subprogram bodies declared in the same declarative part.

. C41404A: The values of 'LAST and 'LENGTH are incorrect in the if statements from line 74 to the end of the test.

. B45116A: ARRPRIBL 1 and ARRPRIBL 2 are initialized with a value of the wrong type--PRIBOOL_TYPE instead of ARRPRIBOOL_TYPE--at line 41.

. C48008A: The assumption that evaluation of default initial values occurs when an exception is raised by an allocator is incorrect according to AI-00397.

. B49006A: Object declarations at lines 41 and 50 are terminated incorrectly with colons, and end case; is missing from line 42.

. B4A010C: The object declaration in line 18 follows a subprogram body of the same declarative part.

. B74101B: The begin at line 9 causes a declarative part to be treated as a sequence of statements.

. C87B50A: The call of "/=" at line 31 requires a use clause for package A.

. C92005A: The "/=" for type PACK.BIG_INT at line 40 is not visible without a use clause for the package PACK.

. C940ACA: The assumption that allocated task TT1 will run prior to the main program, and thus assign SPYNUMB the value checked for by the main program, is erroneous.

. CA3005A..D: (4 tests) No valid elaboration order exists for these tests.

. BC3204C: The body of BC3204C0 is missing.

END

FEB.

1988

DTIC